TRANSPORTATION OPTIMIZATION

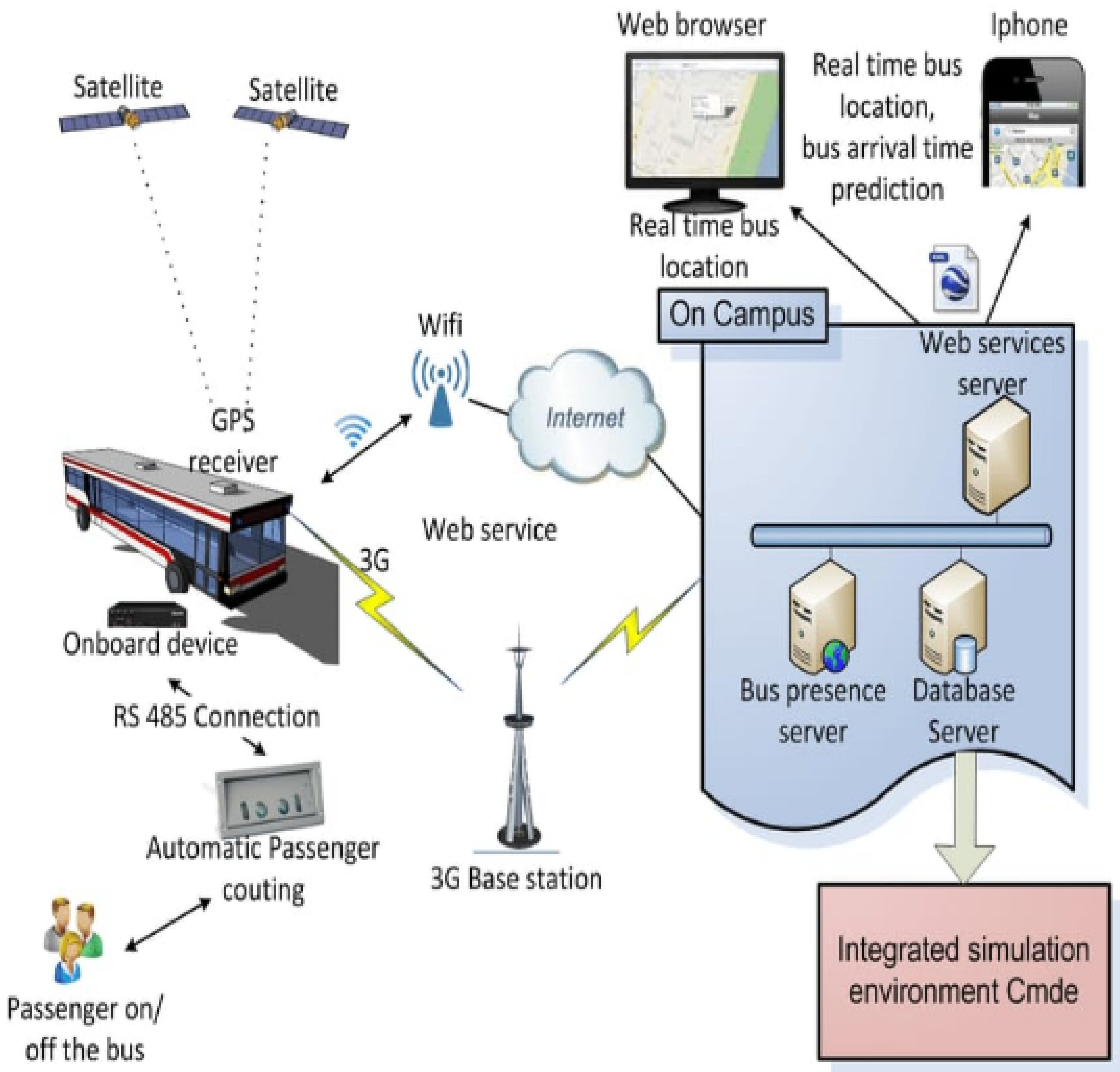
Transportation optimization helps shippers, 3PLs, and transportation consultants analyze shipments, rates, and constraints to produce realistic load plans that reduce overall freight spend.

PCC
PEERLESS CONTENT CREATION

MercuryGate
TMS that delivers.

Transport optimization definition:

TRANSPORTATION OPTIMIZATION is the process of analyzing shipments, rates and Constraints to produce realistic load plans that reduce overall freight spend and gain Efficiencies across entire transportation networks.



Strategies:

WHAT STRATEGIES MAKES SENSE FOR YOUR SHIPMENTS TO BE OPTIMIZED

Smarter | Stronger | Faster | Better

⇨ Consolidation	Aggregating multiple shipments potentially across clients : example LTL to TL
⇨ Multi-point optimization	Build multi stop route with multi pick/ multi drop
⇨ Zone skipping	Consolidate individual packages to TL/LTL and send to distribution/sorting facility closer to destination
⇨ Pooling	Optimal selection of warehouse/cross docks to leverage consolidation and deconsolidation
⇨ Continuous moves	Stringing loads together, so that the carrier can better utilize a particular truck or asset
⇨ Backhauls	Reduce empty miles and increase asset utilization

SENSOR REQUIRED:

***Fuel level measurement sensor**

***Fuel flow meters**

***Torque sensor**

***Air monitoring sensor**

***Speed measurement sensor**

GOALS:

1.Reduced Traffic congestion:

*By helping users find the quickest routes and avoid traffic congestion,machine learning models contribute to reducing overall traffic congestion

In urban areas.

*This can lead to shorter commute times,reduced fuel consumption,and lower emission.

2.Traffic Management

*City traffic management authorities can use machine learning models to optimize traffic signals timings and reduce traffic bottlenecks,improving overall mobility.

3.Cost Efficiency

* Optimize routes to minimize fuel consumption and travel time,which can be critical for logistic and transportation companies looking to reduce operational costs.

4.Privacy and security

*Implement robust privacy measures to protect user data,including their locations,ensure that data is stored and transmitter securely.

ALGORITHM:

STEP 1:Railway traffic control targeting passenger flows

STEP 2:Industrial issues for Mass Transit Operations

STEP 3:Public Transport in Emergency Planning

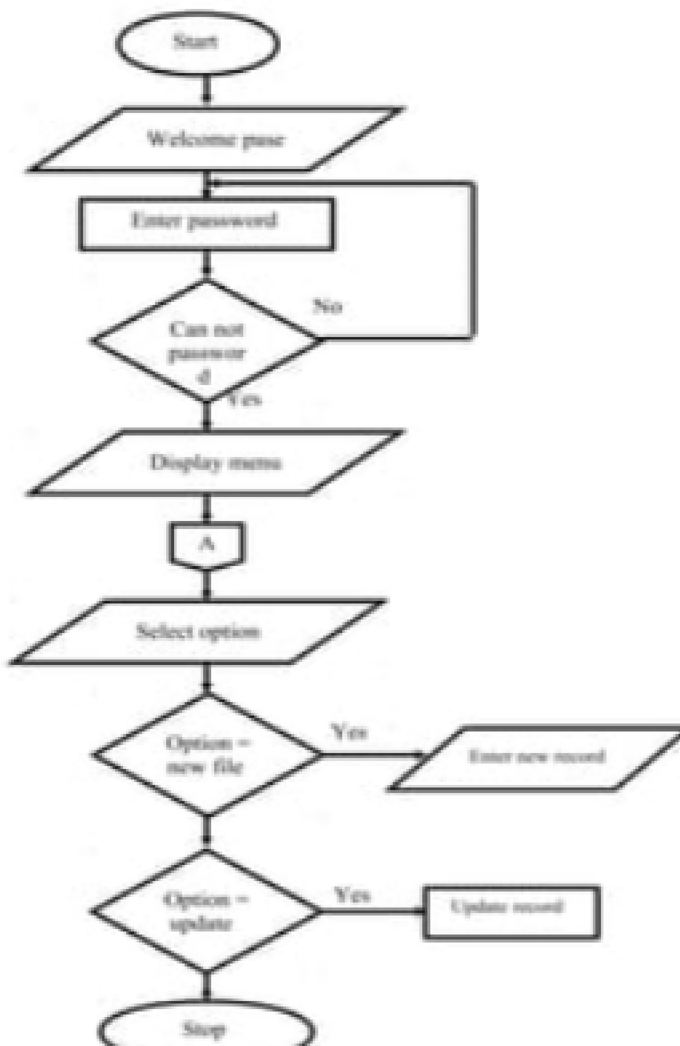
STEP 4:Computing and Improving Passenger Punctuality

STEP 5:Dealing with uncertainty in railway traffic management and Disruption management

STEP 6:Rolling stock planning and challenges

FLOWCHART:

4.9 PROGRAM FLOW CHART



Program 1:

```
# Creates a list of all the supply nodes

Warehouses = ["A", "B", "C"]



# Creates a dictionary for the number of units of supply for each supply node

Supply = {"A": 300, "B": 600, "C":600}



# Creates a list of all demand nodes

Projects = ["1", "2", "3"]



# Creates a dictionary for the number of units of demand for each demand node

Demand = {

    "1": 150,

    "2": 450,

    "3": 900,

}
```

```
# Creates a list of costs of each transportation path

Costs = [  # Projects

    [5,1,9],  # A   warehouses

    [4,2,8],  # B

    [8,7,2]   # C

]


# The cost data is made into a dictionary

Costs = makeDict([warehouses, projects], costs, 0)
```

OUTPUT:

```
Route_A_1 = 0.0

Route_A_2 = 300.0

Route_A_3 = 0.0

Route_B_1 = 150.0

Route_B_2 = 150.0

Route_B_3 = 300.0

Route_C_1 = 0.0

Route_C_2 = 0.0

Route_C_3 = 600.0

Value of Objective Function =  4800.0
```

PROGRAM 2:

```
// C program to find minimum number of platforms required
on

// a railway station

 // Importing the required header files

#include <stdio.h>

 // Creating MACRO for finding the maximum number

#define max(x, y) (((x) > (y)) ? (x) : (y))

 // Function to find the minimum number of platforms

// required

Int findPlatform(int arr[], int dep[], int n)

{

   // plat_needed indicates number of platforms

  // needed at a time

  Int plat_needed = 1, result = 1;

 // Run a nested for-loop to find the overlap

  For (int I = 0; I < n; i++) {

  // Initially one platform is needed

     Plat_needed = 1;
```

```
For (int j = 0; j < n; j++)

If (I != j

// Increment plat_needed when there is an

// overlap

If (arr[i] >= arr[j] && dep[j] >= arr[i])

Plat_needed++;

}

// Update the result

Result = max(plat_needed, result);

}

Return result;

}

// Driver Code

Int main()

{

// Train 1 => Arrival : 01:00, Departure : 09:00

// Train 2 => Arrival : 03:00, Departure : 04:00

// Train 3 => Arrival : 05:00, Departure : 06:00

Int arr[] = { 100, 300, 500 };

Int dep[] = { 900, 400, 600 };
```

```
Return 0;

}
```

OUTPUT:

2

PROGRAM 3:

```
// C++ program to implement the above approach

#include <bits/stdc++.h>

Using namespace std;

// Function to find the minimum number

// of platforms required

Int findPlatform(int arr[], int dep[], int n)

{

    // Store the arrival and departure time

    Vector<pair<int, int> > arr2(n);

    For (int I = 0; I < n; i++) {

        Arr2[i] = { arr[i], dep[i] };

}
```

```
    // Sort arr2 based on arrival time

    Sort(arr2.begin(), arr2.end());

    Priority_queue<int, vector<int>, greater<int> > p;

    Int count = 1;

    p.push(arr2[0].second);

    for (int I = 1; I < n; i++) {

        // Check if arrival time of current train

        // is less than or equals to departure time

        // of previous train

        If (p.top() >= arr2[i].first) {

            Count++;

        }

        Else {

            p.pop();

        }

        p.push(arr2[i].second);

    }

    // Return the number of trains required

    Return count;

}

// Driver Code
```

```
    // Return the number of trains required

    Return count;

}

// Driver Code

Int main()

{

    Int arr[] = { 900, 940, 950, 1100, 1500, 1800 };

    Int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };

    Int n = sizeof(arr) / sizeof(arr[0]);

    Cout << findPlatform(arr, dep, n);

    Return 0;

}
```

OUTPUT:

3


thank
you